BEFORE THE UNITED STATES PATENT AND TRADEMARK OFFICE
BOARD OF APPEALS AND INTERFERENCES

In re Application of:	)	
	)	Group Art Unit: 2162
PAUL E. McKENNEY ET AL.	)	
	)	
SERIAL NO.: 10/795941	)	Confirmation No.: 1342
	)	
FILED: March 8, 2004	)	
	)	Examiner: Fred A. Ehichioya
FOR: EFFICIENT SUPPORT OF CONSISTENT CYCLIC SEARCH WITH READ-COPY UPDATE	)	

Commissioner for Patents
P. O. Box 1450
Alexandria, VA 22313-1450

Sir:

CORRECTED BRIEF ON APPEAL

This is an appeal from the Primary Examiner of Group Art Unit 2162 refusing claims 1-31 set forth in the attached CLAIMS APPENDIX. The date of the final rejection is June 18, 2007 and the Notice of Appeal was filed on November 18, 2007.

REAL PARTY IN INTEREST

International Business Machines Corporation

RELATED APPEALS AND INTERFERENCES

None.

STATUS OF THE CLAIMS

Claims 1-31 are in the application.

Claims 1-31 are rejected.

Claims 1-31 are appealed.

STATUS OF AMENDMENTS

No amendments have been made subsequent to the final rejection.

SUMMARY OF THE CLAIMED SUBJECT MATTER

Introduction

The present invention relates to computer systems and methods in which data resources are shared among concurrent data consumers while preserving data integrity and consistency relative to each consumer (S1:6-8). [1] More particularly, the invention concerns improvements to a mutual exclusion mechanism known as "read-copy update" or "RCU," in which lock-free data read operations run concurrently with data update operations (S1:8-10). The read-copy update technique is well suited to multiprocessor computing environments in which the number of reader processes accessing a shared data set is large in comparison to the number of updater processes, and wherein the overhead cost of employing other mutual exclusion techniques (such as locks) for each read operation would be high (S1:16-19). By

1 As used herein, the citation format "Sx:y" refers to Specification, page x, line y.

way of example, a network routing table that is updated at most once every few minutes but searched many thousands of times per second is a case where read-side lock acquisition would be quite burdensome (S1:19-22).

The read-copy update technique implements data updates in two phases (S2:1). In the first phase, the actual data update is carried out in a manner that temporarily preserves two views of the data being updated (S2:1-3). One view is the old (pre-update) data state that is maintained for the benefit of operations that may be currently referencing the data (S2:3-4). The other view is the new (post-update) data state that is available for the benefit of operations that access the data following the update (S2:4-6). In the second phase, the old data state is removed following a "grace period" that is long enough to ensure that all executing operations will no longer maintain references to the pre-update data (S2:6-8).

In the context of the read-copy update mechanism, a grace period represents the point at which all running processes having access to a data element guarded by read-copy update have passed through a "quiescent state" in which they can no longer maintain references to the data element, assert locks thereon, or make any assumptions about data element state (S3:22-S4:2). By convention, for operating system kernel code paths, a context (process) switch, an idle loop, and user mode execution all represent quiescent states for any given CPU (as do other operations that will not be listed here) (S4:2-5).

There are various methods that may be used to implement a deferred data update following a grace period, including but not limited to the use of callback processing as described in commonly assigned U.S. Patent No. 5,727,209, entitled "Apparatus And Method For Achieving Reduced Overhead Mutual-Exclusion And Maintaining Coherency In A Multiprocessor System Utilizing Execution History And Thread Monitoring" (S4:14-17). The

callback method is also described in the cited NPL prior art reference "Read-Copy Update" by Paul E. McKenney et al. (Mr. McKenney being one of the Applicants herein). A read-copy update callback represents a request by an updater for deferred (second phase) removal of a stale data element. Pending callbacks are accumulated on a callback list until the expiration of a grace period indicating that all of the callbacks can be processed. The callback lists are used for batch processing of the accumulated callbacks. Typically, there are callback lists for multiple grace periods, including one list for new callbacks accumulated during the current grace period, a second list for callbacks accumulated during the immediately preceding grace period, and a third list for callbacks accumulated two grace periods ago. The callbacks on the third list represent the oldest callbacks and are processed first. When this occurs, the callbacks on the first and second lists are respectively advanced to the second and third lists. The first callback list is then available to be populated with new callbacks.

A number of variants of read-copy update have been used in different operating systems (S4:20). However, all of these implementations make at least one of the following assumptions:

1) Stale data is permissible (for example, in read-copy update-protected routing tables) (S4:23).

2) Readers search the aggregate data structure in an acyclic manner, so that there is no possibility of a reading process seeing two different versions of the same data element during a single operation. This assumption also implies that, for data element groups having multiple entry points, a given search starts with only one of these entry points (S5:1-4).

3) There is no need for multiple data elements to be seen in a consistent aggregate state. Consistency is important only for a given data element (as, for example, the data structures used in the Linux 2.6 kernel's read-copy update-protected System V IPC (InterProcess Communication) mechanism) (S5:5-8).

4) If group consistency is important for a collection of data elements, read-copy update must be used in a manner that allows the group to be updated atomically so as to protect group integrity. As used herein, the term "atomic" signifies that the data update operation must complete with the guarantee that no other process will see inconsistent versions of the group data elements. For example, in the Linux 2.6 kernel, the directory-cache is protected by read-copy update, but per-entry locks are also used to ensure that updates to these entries and their associated inodes are in a coordinated consistent state when cache readers access the entries. Another approach would be to make a copy of the aggregate data structure (i.e., the entire collection of data elements), update the new copy, and then link the new copy in place of the old copy. However, this is extremely time consuming for large groups, and is particularly inefficient when only small changes are required (S5:9-20).

Cyclic searches represent a situation where none of the foregoing assumptions underlying the use of read-copy update are in play (S5:21-22). An example of a commonly used cyclic search is the traversal of a group of data elements representing a finite state machine (S5:22-23). If these data elements change dynamically, but infrequently, in comparison to the number of read traversals, then the use of read-copy update could be advantageous (S5:23-S6:2). However, it will be seen that:

1) Permitting stale data could result in a reader seeing an inconsistent, and possibly nonsensical, finite state machine (S6:4-5).

2) Traversing a finite state machine is in general an inherently cyclic activity (S6:6).

3) Each reader must see a finite state machine that is consistent as a whole – consistency of a particular state is not sufficient (S6:7-8).

4) If the finite state machine is large, implementing atomic data element group updates by group copying will be infeasible (S6:9-10).

The present invention allows read-copy update to be used when there is a group of data elements that must be updated atomically as a group within a single update operation, so that readers are presented with a consistent view of the data during any given read operation (S9:13-15). Fig. 5 of the drawings illustrates an example of such a data group in the form of a cyclic graph 20 comprising a linked set of data elements A, B and C (D:5; S9:15-17). These linked data elements can be constituted using any suitable programming construct, including but not limited to C-language "struct" (data structure) variables comprising one or more data fields and one or more pointers to other elements in the data group (S9:17-20). During a cyclic search (read operation) involving the graph 20, a reader enters the graph by following the global pointer to A (S9:20-22). Depending on the processing result generated at A, the reader may either move to B and thence to C, or directly to C (S9:22-23). From C, the reader returns to A (S9:23). It will be seen that if any of the data elements A, B or C is modified or deleted using the conventional read-copy update mechanism, readers will be presented with an inconsistency if they reencounter the data element or find that it has been removed during the same read operation (S9:23-S10:3). For example, if the graph 20 is a state machine, the logic of a reader encountering the updated data element may be thrown into disarray, possibly

causing the reader to take an erroneous action or crash (S10:3-5). An inconsistency problem could also occur if a new data element is added while a read operation is in progress (S10:5-7).

Attention is now directed to Figs. 6, 7 and 8A-8F of the drawings. Fig. 6 illustrates a generalized method that can be employed in accordance with the present invention to maintain data integrity while updating a shared data element group such as the cyclic graph 20 (S10:8-10) of Fig. 5 that is shown undergoing an update in Figs 8A-8F. In an initialization step 30, a global generation number is established relative to the data element group and each data element in the group is assigned a copy of the global generation number at the time of its creation (S10:10-12). In steps 32 and 34, an updater that wishes to replace, delete or insert a group data element generates a new data element and sets its generation number field to a value that is one greater than the current global generation number (S10:12-15). If the updater is replacing a current data element, the new data element will be a modified copy of the current data element (S10:15-16). If the updater is deleting a current data element, the new data element will be a copy of the current data element with a "deleted" flag set (S10 :16-17). If the updater is inserting a new data element, the new data element is created from scratch (S10:17-18). In step 36, the updater sets version links between the new data element and its pre-update version (if such a version exists) (S10:18-20). In particular, each data element version maintains a set of two version pointers, one being an old-version pointer to a previous version of the data element (if any) and the other being a new-version pointer to a next version of the data element (if any) (S10:20-23). A NULL old-version pointer is used for any data element having no previous version (i.e., the data element has not been updated since the last grace period or is an insert) (S10:23-S11:2). A NULL new-version pointer used for any data

element having no next version (i.e., it is the most current version) (S11:2-3). In step 38, the updater changes any link (list) pointers that point to the old version of the data element to instead point to the new version, and then increments the global generation number (S11:3-5).

Fig. 7 of the drawings illustrates a generalized method that can be employed in accordance with the present invention while searching (reading) a shared data element group such as that shown by the cyclic graph 20 (S11:6-8). In step 40, a reader determines the present value of the global generation number and assigns it to the search (S11:8-9). In step 42, the reader traverses the data element group following the links between data elements (S11:9-10). In step 44, as each data element is read the reader compares the global generation number assigned to the search with the data element's copy of the global generation number to determine if the two generation numbers match (indicating that the data element is valid for this reader) (S11:10-13). If the data element's generation number does not match that of the reader, then in step 46, the reader follows the data element's version pointers to new and old versions of the data element as necessary to find a version that does have a matching generation number, if any (S11:13-16).

Turning now to Figs. 8A-8F of the drawings, an exemplary update of the cyclic graph 20 of Fig. 5 will now be described to illustrate how the methods of Figs. 6 and 7 can be implemented in a practical application (S11:17-19). In Fig. 8A, the cyclic graph 20 is shown with the data elements A, B and C being in an initial state (S11:19-20). The subscripts associated with each data element indicate generation number (S11:20-21). The global generation number is illustrated by the circled number on the left-hand side of each figure (S11:21-22). It is assumed that the graph 20 is to be updated by replacing data elements A and C with new versions (S11:22-23).

In Fig. 8B, two replacement data elements A_2 and C_2 are allocated, their corresponding fields are copied from A_1 and C_1, respectively, and the old/new-version pointers of the old and new data element versions are filled in (S12:1-3). The arrows with the circle represent old-version pointers, and the arrows with the diamond represent new-version pointers (S12:3-5). Version pointers having NULL values are not shown (S12:5). Note that any first generation reader that enters the graph 20 at this point will proceed unaware of the new data elements because the version numbers of the data elements traversed will match the current search generation number (S12:5-8). Such readers will not attempt to traverse the new-element pointers from A_1 and C_1 to A_2 and C_2 because there is no need to do so (S12:8-9).

As shown in Fig. 8C, the next step is to execute any required memory-barrier instructions (which may be necessary for CPUs with weak memory consistency models) and then start changing data element link (list) pointers to link the new data elements into the data element group (S12:10-13). The first pointer that can be changed is the one emanating from B_1 (although the pointers could be updated in any order), so that B_1 is now linked to C_2 (S12:13-14). At this point, first generation readers will start encountering C_2 (S12:14-15). However, according to the search method outlined above, such readers will note the generation number mismatch, and follow the old-version pointer from C_2 to C_1 (S12:15-17).

As shown in Fig. 8D, the next pointer to be updated is the graph's global pointer that previously pointed to A_1 (see Fig. 8C) and will now point to A_2 (S12:18-19). Once the global pointer is updated, first generation readers will start encountering both A_2 and C_2, and will follow the old-version pointers upon noting the generation number mismatch (S12:19-21).

As shown in Fig. 8E, the updater completes the update by incrementing the global generation number (S12:22-23). However, assuming no memory-barrier instructions are used

at this point, readers might see the global generation number update and the global pointer update in any order (S12:23-S13:2). The old/new-version pointers will direct the readers to the correct data element version in either case (S13:2-3). Although the use of memory barriers could allow some simplification of the search process in some special cases, the foregoing methodology has the advantage of generality (S13:3-5). Note that second generation readers will still find B_1, but will accept it as the current version because B_1 has no new-version pointer (S13:5-6).

As shown in Fig. 8F, once a grace period has elapsed, old data elements A_1 and C_1 may be discarded, along with their corresponding link pointers (S13:7-8). The old/new-version pointers for all remaining data elements are set to NULL (as necessary) (S13:8-9).

A modified version of the foregoing method is shown in Figs. 14A-14E of the drawings. This embodiment is advantageous if it may be necessary to maintain multiple sets of old/new version pointers, potentially one set for each link pointer that points to a data element (S18:14-15). A simple way of handling this in some cases will be to use "pointer-forwarding" entities, similar to those used by some dynamic linkers (S18:15-17). The pointer forwarding entities can be implemented as a C-language "struct" variables or using any other suitable programming construct (S18:17-18). Each such entity will contain a link pointer to an actual data element, a snapshot of the global generation number, and pointers to old and new versions of that pointer-forwarding entity (S18:18-20). The pointer-forwarding entity may also contain pointers used to locate other pointer-forwarding entities referencing the same data element or referenced by the same data element, as well as a pointer to the data element referencing this pointer-forwarding entity (S18:20-24). In contrast to the method described above wherein version links are maintained between data elements, the modified method

maintains version links between the pointer-forwarding entities. Details can be found at S19:6-S20:2 and D13, and additionally at S20:3-S21:14 and D14A-14E.

Independent Claim 1

Claim 1 is a method claim directed to "updating a shared data element group [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions" (C1 :preamble).

The first paragraph of the body of claim 1 recites "generating a new group data element." Support for this limitation is found at S10:12-15, D6:32 and D8B:A_2 and C_2.

The second paragraph of the body of claim 1 recites "assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader." Support for this limitation is found at S10:12-15 and D6:34.

The third paragraph of the body of claim 1 recites "if a prior version of said new data element exists, establishing a version link between said new data element and said prior version." Support for this limitation is found at S10:18-20, D6:36 and D8B.

The fourth paragraph of the body of claim 1 recites "linking said new data element into said data element group so that it is reachable by readers." Support for this limitation is found at S11:3-5, D6:38, D8C:C_2 and D8D:A_2.

The fifth paragraph of the body of claim 1 recites "updating a global generation number associated with said data element group." Support for this limitation is found at S11:3-5, D6:38 and D8E.

The sixth paragraph of the body of claim 1 recites "if a prior version of said new data element exists, freeing said prior version following a grace period." Support for this limitation is found at S13:7-9 and D8F.

Independent Claim 6

Claim 6 is a method claim directed to "updating a shared data element group [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, without using locks or atomic instructions" (C6:preamble).

The first paragraph of the body of claim 6 recites "generating a pointer-forwarding entity that points to a data element in said data element group." Support for this limitation is found at S19:11-13, D13:146 and D14B:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The second paragraph of the body of claim 6 recites "assigning a generation number to said pointer-forwarding entity that allows a reader of said data element group to determine whether said pointer-forwarding entity is a correct version for said reader." Support for this limitation is found at S19:13-16, and D13:148.

The third paragraph of the body of claim 6 recites "if there is a prior version of said pointer-forwarding entity, establishing a version link between said pointer-forwarding entity and said prior version." Support for this limitation is found at S19:13-16, D13:148 and D14B.

The fourth paragraph of the body of claim 6 recites "linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity." Support for this limitation is found at S19:19-20, D13:154 and D14C:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The fifth paragraph of the body of claim 6 recites "updating a global generation number associated with said data element group." Support for this limitation is found at S19:23-S20:1, D13:156 and D14D.

The sixth paragraph of the body of claim 6 recites "if a prior version of said pointer-forwarding entity exists, freeing said prior version following a grace period." Support for this limitation is found at S20:1-2, D13:158 and 14E.

Dependent Claim 7 [2]

Claim 7 is a method claim directed to "performing a search of a shared data element group that may be undergoing modification in accordance with the method of claim 1 [such as the group 20 of Figs. 8A-8F]" (C7:preamble).

The first paragraph of the body of claim 7 recites "assigning a current global generation number to said search." Support for this limitation is found at S11:6-9 and D7:40.

The second paragraph of the body of claim 7 recites "when referencing a data element in said data element group, determining whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number." Support for this limitation is found at S11:10-13 and D7:44.

The third paragraph of the body of claim 7 recites "searching for a correct version of said referenced data element as necessary." Support for this limitation is found at S11:13-16, D7:46.

Independent Claim 11

Claim 11 is a system claim directed to "a data processing system having one or more central processing units [D4:4], a memory [D4:8] and a communication pathway [D4:6]

between the one or more central processing units and the memory, said system being adapted to update a shared data element group [D4:16] in said memory [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers [D4:18_2-18_n] that are concurrently referencing group data elements without using locks or atomic instructions" (C11:preamble).

The first paragraph of the body of claim 11 recites "means for generating a new group data element." Support for this limitation is found at S10:12-15, D4:18_1, D6:32 and D8A:A_2.

The second paragraph of the body of claim 11 recites "means for assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader." Support for this limitation is found at S10:12-15, D4:18_1 and D6:34.

The third paragraph of the body of claim 11 recites "means for establishing, if a prior version of said new data element exists, a version link between said new data element and said prior version." Support for this limitation is found at S10:18-20, D4:18_1, D6:36 and D8B.

The fourth paragraph of the body of claim 11 recites "means for linking said new data element into said data element group so that it is reachable by readers." Support for this limitation is found at S11:3-5, D4:18_1, D6:38, D8C:C_2 and D8D:A_2.

The fifth paragraph of the body of claim 11 recites "means for updating a global generation number associated with said data element group." Support for this limitation is found at S11:3-5, D4:18_1, D6:38 and D8E.

---

2 Although claim 7 is dependent and does not contain means plus function elements, cross-references to the specification are nonetheless being provided for convenience.

The sixth paragraph of the body of claim 11 recites "means for freeing, if a prior version of said new data element exists, said prior version following a grace period." Support for this limitation is found at S13:7-9, D4:18_1 and D8F.

Dependent Claim 15

Dependent claim is system claim directed to "a system in accordance with claim 11 wherein said system further includes means for generating a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new data element and further being used to link said new data element into said data element group." Support for the means plus function element is found at S19:11-21, D4:18_1, D13:146-154, D14B:$P(A)_2$, $P(B)_2$ and $P(C)_2$, and D14C:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

Independent Claim 16

Claim 16 is a system claim directed to "a data processing system having one or more central processing units [D4:4], a memory [D4:8] and a communication pathway [D4:6] between the one or more central processing units and the memory, said system being adapted to update a shared data element group [D4:16] in said memory [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers [D4:18_2-18_n] that are concurrently referencing group data elements without using locks or atomic instructions" (C16:preamble).

The first paragraph of the body of claim 16 recites "means for generating a pointer-forwarding entity that points to a data element in said data element group." Support for this limitation is found at S19:11-13, D4:18_1, D13:146 and D14B:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The second paragraph of the body of claim 16 recites "means for assigning a generation number to said pointer-forwarding entity that allows a reader of said data element

group to determine whether said pointer-forwarding entity is a correct version for said reader." Support for this limitation is found at S19:13-16, D4:18_1 and D13:148.

The third paragraph of the body of claim 16 recites "means for establishing, if there is a prior version of said pointer-forwarding entity, a version link between said pointer-forwarding entity and said prior version." Support for this limitation is found at S19:13-16, D4:18_1, D13:148 and D14B.

The fourth paragraph of the body of claim 16 recites "means for linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity." Support for this limitation is found at S19:19-20, D4:18_1, D13:154 and D14C:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The fifth paragraph of the body of claim 16 recites "means for updating a global generation number associated with said data element group." Support for this limitation is found at S19:23-S20:1, D4:18_1, D13:156 and D14D.

The sixth paragraph of the body of claim 16 recites "means for freeing, if a prior version of said pointer-forwarding entity exists, said prior version following a grace period." Support for this limitation is found at S20:1-2, D4:18_1, D13:158 and 14E.

Dependent Claim 17

Claim 7 is system claim directed to "a data processing system having one or more central processing units [D4:4], a memory [D4:8] and a communication pathway [D4:6] between the one or more central processing units and the memory, said system being adapted to perform a search of a shared data element group that may be undergoing modification in

accordance with the system of claim 11 [such as the group 20 of Figs. 8A-8F]" (C17:preamble).

The first paragraph of the body of claim 17 recites "means for assigning a current global generation number to said search." Support for this limitation is found at S11:6-9, D4:18_2-18_n and D7:40.

The second paragraph of the body of claim 17 recites "means for determining, when referencing a data element in said data element group, whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number." Support for this limitation is found at S11:10-13, D4:18_2-18_n and D7:44.

The third paragraph of the body of claim 17 recites "means for searching for a correct version of said referenced data element as necessary." Support for this limitation is found at S11:13-16, D4:18_2-18_n) and D7:46.

Independent Claim 21

Claim 21 is "a computer program product claim directed to updating a shared data element group [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions" (C21:preamble).

The first paragraph of the body of claim 21 recites "one or more data storage media." Support for this limitation is found at D4:8.

The second paragraph of the body of claim 21 recites "means recorded on said data storage medium for programming a data processing platform to operate." Support for this limitation is found at D4:18_1 and D6.

The third paragraph of the body of claim 21 recites "generating a new group data element." Support for this limitation is found at S10:12-15, D6:32 and D8B:A_2 and C_2.

The fourth paragraph of the body of claim 21 recites "assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader." Support for this limitation is found at S10:12-15 and D6:34.

The fifth paragraph of the body of claim 21 recites "if a prior version of said new data element exists, establishing a version link between said new data element and said prior version." Support for this limitation is found at S10:18-20, D6:36 and D8B.

The sixth paragraph of the body of claim 21 recites "linking said new data element into said data element group so that it is reachable by readers." Support for this limitation is found at S11:3-5, D6:38, D8C:C_2 and D8D:A_2.

The seventh paragraph of the body of claim 21 recites "updating a global generation number associated with said data element group." Support for this limitation is found at S11:3-5, D6:38 and D8E.

The eighth paragraph of the body of claim 21 recites "if a prior version of said new data element exists, freeing said prior version following a grace period." Support for this limitation is found at S13:7-9 and D8F.

Dependent Claim 25

Claim 25 is "a computer program product in accordance with claim 21 wherein said means recorded on said data storage media are further adapted to generate a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new data element and further being used to

link said new data element into said data element group." Support for the means plus function element is found at S19:11-21, D4:18_1, D13:146-154, D14B:$P(A)_2$, $P(B)_2$ and $P(C)_2$, D14C:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

Independent Claim 26

Claim 26 is "a computer program product claim directed to updating a shared data element group [such as the group 20 of Figs. 8A-8F] while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions" (C26:preamble).

The first paragraph of the body of claim 26 recites "one or more data storage media." Support for this limitation is found at D4:8.

The second paragraph of the body of claim 26 recites "means recorded on said data storage medium for programming a data processing platform to operate." Support for this limitation is found at D4:18_1 and D13.

The third paragraph of the body of claim 26 recites "generating a pointer-forwarding entity that points to a data element in said data element group." Support for this limitation is found at S19:11-13, D13:146 and D14B:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The fourth paragraph of the body of claim 26 recites "assigning a generation number to said pointer-forwarding entity that allows a reader of said data element group to determine whether said pointer-forwarding entity is a correct version for said reader." Support for this limitation is found at S19:13-16, and D13:148.

The fifth paragraph of the body of claim 26 recites "if there is a prior version of said pointer-forwarding entity, establishing a version link between said pointer-forwarding entity and said prior version." Support for this limitation is found at S19:13-16, D13:148 and D14B.

The sixth paragraph of the body of claim 26 recites "linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity." Support for this limitation is found at S19:19-20, D13:154 and D14C:$P(A)_2$, $P(B)_2$ and $P(C)_2$.

The seventh paragraph of the body of claim 26 recites "updating a global generation number associated with said data element group." Support for this limitation is found at S19:23-S20:1, D13:156 and D14D.

The eighth paragraph of the body of claim 26 recites "if a prior version of said pointer-forwarding entity exists, freeing said prior version following a grace period." Support for this limitation is found at S20:1-2, D13:158 and 14E.

Dependent Claim 27

Claim 27 is "a computer program product claim directed to performing a search of a shared data element group that may be undergoing modification in accordance with the method of claim 1 [such as the group 20 of Figs. 8A-8F]." (C27:preamble).

The first paragraph of the body of claim 27 recites "one or more data storage media." Support for this limitation is found at D4:8.

The second paragraph of the body of claim 27 recites "means recorded on said data storage medium for programming a data processing platform to operate." Support for this limitation is found at D4:18_2-18_n and D7.

The third paragraph of the body of claim 27 recites "assigning a current global generation number to said search." Support for this limitation is found at S11:6-9 and D7:40.

The fourth paragraph of the body of claim 27 recites "when referencing a data element in said data element group, determining whether said referenced data element is a correct

version by comparing a generation number assigned to said referenced data element with said search generation number." Support for this limitation is found at S11:10-13 and D7:44.

The fifth paragraph of the body of claim 27 recites "searching for a correct version of said referenced data element as necessary." Support for this limitation is found at S11:13-16, D7:46.

Independent Claim 31

Claim 31 is "a computer program product claim directed to managing a shared data element group [such as the group 20 of Figs. 8A-8F] so as to allow updates thereof while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions" (C31:preamble).

The first paragraph of the body of claim 31 recites "one or more data storage media." Support for this limitation is found at D4:8.

The second paragraph of the body of claim 31 recites "means rec orded on said data storage medium for programming a data processing platform to operate." Support for this limitation is found at D4:18_1, D4: 18_2-18_n, D6 and D7.

The third paragraph of the body of claim 31 recites "performing a first-phase update operation that preserves a consistent pre-update view of said data element group on behalf of pre-update readers and a consistent post-update view of the data element group on behalf of post-update readers." Support for this limitation is found at S10:12-S11:5, D6:32-38 and D8A-D8E.

The fourth paragraph of the body of claim 31 recites "providing means by which readers can locate all data elements of said data element group that belong to each of said pre-

update and post-update views as readers search said data element group." Support for this limitation is found at S10:18-20, D6:36 and D8B.

The fifth paragraph of the body of claim 31 recites "performing one or more read operations following said first-phase update operation in which one or more readers search said data element group with each reader referencing only data elements belonging to one of said pre-update and post-update views." Support for this limitation is found at S11:6-S11:16 and D7:40-46.

The sixth paragraph of the body of claim 31 recites "performing a second-phase update operation following a grace period that frees said pre-update view of said data element group." Support for this limitation is found at S13:7-9 and D8F.

GROUNDS FOR REJECTION TO BE REVIEWED ON APPEAL

Paragraph 7 of the final rejection states that claims 1-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Applicant Admitted Prior Art ("APA") S1-S6 and D1A-D3 in view of non-patent literature ("NPL") "Read-Copy Update" by Paul E. McKenney et al. "McKenney." The obviousness rejection of paragraph 7 also relies on, but does not cite in the statement of rejection, a newly added third reference, which is U.S. Patent No. 6,886,162 of Paul E. McKenney. Paragraph 8 of the final rejection states that claim 31 is rejected under 35 U.S.C. 102(a) as being anticipated by the APA referred to in the paragraph 7 obviousness rejection. Applicants request review of both grounds of rejection.

ARGUMENT

Procedural Error

As mentioned above, the obviousness rejection of paragraph 7 of the final rejection relies on, but does not cite, the McKenney '162 patent. The '162 patent appears for the first

time in the final rejection in response to Applicants' previous traversal (without claim amendments) of an initial obvious rejection dated February 26, 2007. This initial obviousness rejection was identical to the paragraph 7 final rejection except that the '162 patent was not mentioned. The '162 patent was cited in the paragraph 7 final rejection to bolster the teachings of McKenney NPL by attempting to equate read-copy update callbacks (representing requests for second phase removal of stale data elements) with the group data elements that are referenced by readers according to the rejected claims. Applicants' representative complained to the Examiner and the Examiner's supervisor that the reliance on the '162 constituted a new ground of rejection that was not necessitated by any amendment made by Applicants. Applicants' representative advised that the Examiner had substantially changed the focus of the rejection with the new "callback = data element" theory and that Applicants should be given an opportunity to respond on the merits. Applicants' representative pointed out that the Examiner should have stated in the first Office Action that he was equating the callbacks discussed in the McKenney NPL reference with the claimed data elements. See MPEP § 706.02(j) ("It is important for an examiner to properly communicate the basis for a rejection so that the issues can be identified early and the applicant can be given fair opportunity to reply."). Applicant further advised that it was improper to rely on the '162 patent without citing it in the statement of rejection.

In an examiner interview conducted on August 8, 2007, the examiner advised that the Office Action containing the final rejection would be withdrawn and a new Office Action would be subsequently issued (Interview Summary dated 8/23/07). Despite this proclamation, no new Office Action was forthcoming and Applicants' representative again spoke to the Examiner on 9/12/07, one week before the deadline for filing a notice of appeal. The

Examiner advised that he would prepare the new Office Action before the appeal deadline. He further suggested, for the first time, that the new Office Action might not be non-final (Applicants' Supplemental Interview Summary dated 9/12/07). This would mean that Applicants would again have no opportunity to respond to the newly mentioned '162 patent as a matter of right. To this day, no new Office Action has been received by Applicants or posted to the public PAIR system.

Applicants' request that the final rejection be reversed as being in violation of MPEP § 706.07(a) because it is prematurely final. Section 706.07(a) states that:

> "Under present practice, second or any subsequent actions on the merits shall be final, except where the examiner introduces a new ground of rejection that is neither necessitated by applicant' s amendment of the claims nor based on information submitted in an information disclosure statement filed during the period set forth in 37 CFR 1.97(c) with the fee set forth in 37 CFR 1.17(p)."

The final rejection violates this principle because it effectively makes a new ground of rejection that is based on the '162 patent but which is not necessitated by any amendment.

Alternatively, if the Examiner asserts that the '162 patent is not part of the paragraph 7 final rejection, such that no new ground for rejection has been made, Applicants requests that the reference to the '162 patent be stricken from the record and not considered in determining whether the Examiner has established a prima facie case of obviousness. See MPEP § 706.02(j), which states:

> "Where a reference is relied on to support a rejection, whether or not in a minor capacity, that reference should be positively included in the statement of the rejection. See In re Hoch, 428 F.2d 1341, 1342 n.3 166 USPQ 406, 407 n. 3 (CCPA 1970)."

Should the Board wish to consider additional authority, Applicants direct attention to the non-precedential decision of *Ex parte Robert C. Wohlsen and Sue McNeill*, Appeal No. 2005-0743, Application Serial No. 09/351,723 (2005), for further guidance on the non-consideration of references that are not properly cited in a statement of rejection.

Rejection of Claims 1-30 under 35 U.S.C. 103(a)

As previously stated, claims 1-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Applicant Admitted Prior Art ("APA") S1-S6 and D1A-D3 in view of non-patent literature ('NPL') 'Read-Copy Update" by Paul E. McKenney et al. 'McKenney." Applicants request reversal on the ground that the Examiner has failed to establish prima facie obviousness under section 103.

Discussion of Cited Prior Art

The APA discusses conventional read-copy update as it existed prior to Applicants' invention. As discussed above in the 'Introduction" portion of the section entitled 'Summary Of The Claimed Subject Matter," conventional read-copy update is not suitable for maintaining group integrity in a shared data element group, such as state machines and other group entities subject to cyclic searches. As stated at S5:9-11, group consistency requires that read-copy update be used in a manner that allows the group to be updated atomically so as to protect group integrity. As stated at S5:13-20, this is not possible using conventional read-copy update unless unduly burdensome mechanisms are used to protect readers during updates, such as per-element read locks or copying of the entire group.

The McKenney NPL reference is also directed to conventional read-copy update. Sections 6.1 and 6.2 are relied on in the paragraph 7 final rejection. These sections deal with the use of read-copy update when there are readers that can block as well as be preempted

(which requires special handling to identify read-copy update quiescent states). Section 6.1 discusses grace period generation tracking for conventional read-copy update callback processing using per-CPU counter pairs and per-CPU generation sequence numbers. This section describes how readers that are subject to blocking or preemption can protect themselves from premature grace period termination by incrementing and decrementing per-CPU counters when they enter and leave RCU critical sections, respectively, and how these counters are further used by grace period detection logic to determine when the readers have entered a quiescent state in which they are no longer referencing RCU-protected data. Section 6.2 discusses the use of callbacks and callback lists to batch-schedule the second-phase update portion of read-copy update wherein old data elements are freed following the conclusion of each grace period generation. Three callback lists referred to as "next," "current" and "intr" are identified. The "intr" callback list represents the oldest list whose callbacks will be processed as soon as a new grace period starts. The "current" callback list is the second oldest list. The "next" list is the most recent list that is used for accumulated the latest callbacks.

Legal Framework for Obviousness Analysis

Section 103 requires the issuance of a patent unless "the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains." *KSR Int'l Co. v. Teleflex Inc.*, 127 S.Ct. 1727, 1734, 82 USPQ2d 1385, 1391 (2007). The question of obviousness is resolved on the basis of underlying factual determinations including (1) the scope and content of the prior art, (2) any differences between the claimed subject matter and the prior art, (3) the level of skill in the art, and (4) where in evidence, so-called secondary considerations.

Graham v. John Deere Co., 383 U.S. 1, 17-18, 148 USPQ 459, 467 (1966). See also *KSR*, 127 S.Ct. at 1734, 82 USPQ2d at 1391 ("While the sequence of these questions might be reordered in any particular case, the [Graham] factors continue to define the inquiry that controls.")

The USPTO bears the initial burden of establishing that a claimed invention is prima facie obvious. *In re Piasecki*, 745 F.2d 1468, 1472, 223 USPQ 785, 788 (Fed. Cir. 1984). To establish a prima facie case of obviousness, the USPTO must satisfy three requirements. First, it must "identify a reason that would have prompted a person of ordinary skill in the relevant art to combine the elements in the way the claimed new invention does." *KSR Int'l Co. v. Teleflex Inc.*, *supra*. Second, the proposed modification of the prior art must have had a reasonable expectation of success, determined from the vantage point of the artisan at the time the invention was made. *Amgen, Inc. v. Chugai Pharm. Co.*, 927 F.2d 1200, 1209, 18 USPQ2d 1016, 1023 (Fed. Cir. 1991). Third, the prior art reference or combination of references must teach or suggest all the limitations of the claims. *In re Wilson*, 424 F.2d 1382, 1385, 165 USPQ 494, 496 (CCPA 1970).

On October 10, 2007, the USPTO issued new examination guidelines for determining obviousness under 35 U.S.C. 103 in view of the *KSR* case. Federal Register, Vol. 72, No. 195, pages 57526-57535. The examination guidelines emphasize that "the focus when making a determination of obviousness should be on what a person of ordinary skill in the pertinent art would have known at the time of the invention, and on what such a person would have reasonably expected to have been able to do in view of that knowledge." Fed. Reg., Vol, 72, No. 195 at page 57527. The examination guidelines point out that "[i]n certain circumstances, it may also be important to include explicit findings as to how a person of

ordinary skill would have understood prior art teachings, or what a person of ordinary skill would have known or could have done."

Discussion of Grounds for Rejection to be Reviewed on Appeal

In the discussion to follow, Applicants present separate patentability arguments relative to the following six (8) claim groups that are subject to the paragraph 7 obviousness rejection: (1) independent claims 1, 11 and 21, (2) independent claims 6, 16 and 26, (3) dependent claims 7, 17 and 27, (4) dependent claims 3, 13 and 23, (5) dependent claims 5, 15 and 25, (6) dependent claims 8, 18 and 28, (7) dependent claims 9, 19 and 29, and (8) dependent claims 10, 20 and 30. Applicants additionally present a separate patentability argument relative to claim 31, which is subject to the paragraph 8 anticipation rejection.

Argument Points Applicable To Independent Claims 1, 11 and 21

Relative to independent claims 1, 11 and 21, the Examiner argues that certain sections of the APA discloses several claim elements that are in fact not present. These missing claim elements include (1) the claim preamble wherein the claimed subject matter is said to be directed to 'updating a shared data element group while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions" and (2) the paragraph in the claim body reciting 'establishing a version link between said new data element and said prior version."

Relative to missing claim element (1), the Examiner cites APA at S1:12-16. However, this passage is merely a statement that read-copy update is a mutual exclusion technique that allows shared data to be accessed for reading without expensive synchronization mechanisms, such as locks and enumerated techniques, while still permitting the data to be updated concurrently. The cited passage contains no mention of maintaining group integrity, which, as

discussed in the APA at S5:9-20, requires that group updates be implemented atomically. The only mention in the APA regarding read-copy update and maintaining group integrity is the statement of the problem to be solved and the discussion of the two unsatisfactory prior art methods that require read-locking of data elements during updates or copying an entire data element group (S5:9-20).

Relative to missing claim element (2), the APA does not in fact disclose establishing a version link between said new data element and said prior version. The Examiner cites S3:5-10 as support for his contention that the APA does show version links. However, S3:5-10 is part of a discussion of Figs. 1A-1D, which shows a data element "B" being updated using conventional read-copy update. S3:5-10 specifically discusses how the original element B retains its linked list pointer to C for the benefit of existing readers. The pointers between different data elements are not version links between different versions of the same element, as claimed. There is no discussion or illustration of a version link between the original element B and its updated version B' (see Fig. 1C).To disclose the item (2) claim limitation, there would have to be a linking entity such as a pointer from the new element B' to the old element B, but this is not shown or described insofar as that is part of the claimed invention and not the prior art being shown in Fig. 1C.

The Examiner expressly concedes that the APA does not specifically teach two additional elements of the claims, namely, (3) "assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader," and (4) "updating a global generation number associated with said data element group."

Relative to missing claim elements (3) and (4), the McKenney NPL reference is cited at page 15, column 1, section 6.1; Fig. 29; and pages 16-17, item 5 #3. What the first paragraph of section 6.1 is actually describing is grace period tracking for conventional read-copy update callback processing using per-CPU counter pairs that are manipulated by readers, and per-CPU generation sequence numbers. The generation sequence numbers are not assigned to new data elements (or any other data element) of a data element group per missing claim element (3) so that readers can determine the correct data element version. Nor are they associated with a data element group per missing claim element (4). Rather, the generation sequence numbers are assigned to CPUs and are used to identify which of a pair of per-CPU counters is to be manipulated by the readers. One per-CPU counter is for a current generation of readers and the other per-CPU counter is for a previous generation of readers. When the previous generation counter is reset to zero, the per-CPU generation sequence number is advanced to create a new generation and the roles of the per-CPU counters are reversed. The current generation counter becomes the previous generation counter and the previous generation counter becomes the current counter. After the generation sequence number on a given CPU has been advanced two times, it is deemed (by grace period detection logic) that all read operations in existence prior to the advancement have terminated on that CPU. Section 6.1 of the McKenney NPL reference contains no mention of assigning a generation number to a new data element (or any other data element) per missing claim element (3), or doing so in order to allow a reader of the data element group to determine whether the new data element is a correct version for the reader. Nor does section 6.1 have any bearing on updating a generation number associated with a data element group per missing claim element (4).

Fig. 29 and the passage at pages 16-17, item 5, #3, are also inapposite. Fig. 29 illustrates the callback lists that are used to accumulate callbacks and the data structures that are used to track grace periods and determine when the callbacks may be processed. Page 16, item 5, #3 of the McKenney NPL reference discusses global current-generation counters, maximum generation counters, and per-CPU generation counters, all of which are used to track grace periods for callback processing purposes and none of which are associated with a data element or a data element group.

As earlier mentioned, paragraph 7 of the final rejection also injects the McKenney '162 patent into the discussion. In particular, the '162 patent is used by the Examiner to declare, for the first time during prosecution, that the callbacks discussed in the McKenney NPL reference are actually data elements as recited in claims 1, 11 and 21. However, even assuming the '162 is entitled to consideration as if it had be been properly cited in the statement of rejection, there are several reasons why callbacks would never be interpreted as the claimed data elements by persons of ordinary skill. First, the claimed data elements are recited as being referenced by one or more readers, and this is not the case for callbacks (callbacks are executed, not read). Second, as described in the McKenney NPL at page 15, item #3 of section 6.2, callbacks are not data elements, they are requests containing pointers to callback functions and arguments. Third, callbacks are not assigned generation numbers, they are simply placed on callback lists. Fourth, new callbacks are not created from old callbacks that are maintained until the expiration of a grace period. Fifth, given the fact that there are no multiple versions of the same callback, there can be no version links maintained between new and old callbacks. Sixth, one or ordinary skill familiar with conventional read-copy update would never dream of equating the callback processing subsystem of a read-copy update

implementation with the data elements that RCU is designed to protect. Any such analogy defies common sense.

Based on the foregoing, it is respectfully submitted that independent claims 1, 11 and 21 are patentably distinguishable over the APA and the McKenney NPL reference. The Examiner has failed to present evidence showing that there was any reason for persons of ordinary skill to modify the APA prior art using the McKenney NPL reference or that the claimed invention would have resulted if the references had been combined. Taking the APA and the McKenney NPL reference for all that they teach, and even considering the improperly relied upon McKenney '162 patent, the claimed subject matter simply would not have been obvious to one of ordinary skill in the art. Indeed, the asserted combination of references, even if made, could not have produced the claimed subject matter due to a complete absence of several claim elements. As such, the evidence falls far short of establishing prima facie obviousness relative to claims 1, 11 and 21.

Argument Points Applicable To Independent Claims 6, 16 and 26

The rejection of independent claims 6, 16 and 26 is analogous to the rejection of claims 1, 11 and 21. However, instead of claims referring to data elements, they refer to pointer forwarding entities that point to data elements in a data element group. The rejection of claims 6, 16 and 26 is even more tenuous than the rejection of claims 1, 11 and 21. The rejection not only suffers from all of the errors described in the preceding section, it also fails to show any teaching or suggestion in the cited art of the recited pointer forwarding entities. In particular, the Examiner relies on S3:1-5 as disclosing pointer forwarding entities. This passage discusses Figs. 1A-1D and the creation of a new data element B' from original element B and its insertion into a linked list. There is no mention in the APA of a pointer

forwarding entity. Only data elements and conventional list pointers that link the elements are discussed. A fortiori, there is no disclosure in the APA of maintaining version links between pointer forwarding entities, linking pointer forwarding entities into a data element group, or freeing prior versions of pointer forwarding entities. In the McKenney NPL reference, there is no teaching of assigning generation numbers to pointer forwarding entities. As such, the rejection of claims 6, 16 and 26 should also be reversed.

Argument Points Applicable To Dependent Claims 7, 17 and 27

Relative to dependent claims 7, 17 and 27, the Examiner argues that the McKenney NPL reference discloses the claim elements of (1) "assigning a current global generation number to said search," (2), "determining whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number" "when referencing a data element in said data element group," and (3) "searching for a correct version of said referenced data element as necessary." Each of these claim elements is missing from the McKenney NPL reference.

Relative to missing claim element (1), the McKenney NPL reference is cited at page 16, column 2, item 5, #3. This passage has been discussed above. It is directed to callbacks and callback lists, and the use of various grace period generation counters to track grace periods on CPUs. None of the generation counters mentioned in this passage represents a current global generation number that is assigned to a search. Rather, the counters are used to track grace periods for callback processing.

Relative to missing claim element (2), the McKenney NPL reference is cited at page 15, column 1, paragraph 4. However, as discussed above, the only generation numbers described in this passage are per-CPU generation sequence numbers that are used to track

grace periods. The generation sequence numbers are not assigned to data elements and are not referenced by readers to determine whether a referenced data element is a correct version by comparing the generation number assigned to the data element with a search generation number.

Relative to missing claim element (3), the McKenney NPL reference is cited at page 17, column 1, paragraph 4. However, this passage refers to a CPU advancing its callback lists to facilitate callback processing when a global generation number is advanced. This does not occur as part of searching a data element group and is entirely different from a reader searching for a correct version of a data element.

Argument Points Applicable To Dependent Claims 3, 13 and 23

The references do not disclose or suggest the claimed subject matter wherein the subject matter of claims 1, 11 and 21 is respectively used 'to delete a group data element and said new data element is generated by copying said data element to be deleted and setting a deletion flag in said new data element." The McKenney NPL reference at page 7, column 1, paragraph 4 mentions data element deletion and copying in the same paragraph, but does not disclose copying as part of a deletion operation. The McKenney NPL reference at page 7, column 1, paragraph 3 mentions flagging stale data so that a reader will know the data is stale, but there is no disclosure of a delete flag used for identifying a data element as deleted.

Argument Points Applicable To Dependent Claims 5, 15 and 25

The references do not disclose or suggest the claimed subject matter wherein the subject matter of claims 1, 11 and 25 respectively further includes 'generating a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new data element and further being used to

link said new data element into said data element group." The APA at S3:1-5 does not disclose this feature. Rather, this passage is directed to Fig. 1C and describes how a new data element is created by copying an old data element and linking the new data element into a linked list. There is no disclosure of the claimed pointer forwarding entity.

Argument Points Applicable To Dependent Claims 8, 18 and 28

The references do not disclose or suggest the claimed subject that respectively depends from claims 7, 17 and 27, and "wherein, if said data element generation number is equal to said search generation number, said referenced data element is accepted for reading as a correct version." The McKenney NPL reference at page 17, column 1, paragraphs 3 and 4 is directed to a CPU's manipulation of generation counters to advance a grace period and move callback lists for callback processing. There is no disclosure of accepting a data element for reading if it is a correct version of a referenced data element.

Argument Points Applicable To Dependent Claims 9, 19 and 29

The references do not disclose or suggest the claimed subject matter that respectively depends from claims 7, 17 and 27, and "wherein, if said data element generation number is less than said search generation number, a search is made for a later version of said referenced data element, and wherein said referenced data element is used if a later version is not found." The McKenney NPL reference at page 5, column 1, paragraph 3 is directed to conventional read-copy update searching without locks. There is no disclosure of searching for a later version of a referenced data element if a data element generation number is less than a search generation number.

Argument Points Applicable To Dependent Claims 10, 20 and 30

The references do not disclose or suggest the claimed subject matter that respectively depends from claims 7, 17 and 27, and "wherein, if said data element generation number is greater than said search generation number, a search is made for a prior version of said referenced data element, and wherein said referenced data element is deemed to be a new insertion if there is no prior version. The McKenney NPL reference at page 18, column 1, paragraph 2 is directed to a reader-writer lock that uses per-CPU locks that are cache-aligned for rapid acquisition by readers. There is no disclosure of searching for a prior version of a data element if it has a data element generation number that is greater than a search generation number.

Rejection of Claim 31 under 35 U.S.C. 102(a)

As previously stated, claim 31 was rejected under 35 U.S.C. 102(a) as being anticipated by the APA referred to in the paragraph 7 obviousness rejection. Applicants request reversal on the ground that the Examiner has failed to establish anticipation under section 102.

The APA is said to disclose all of the elements of claim 31. However, there are several claim limitations that are entirely absent in the APA. For example, the APA discloses only a conventional read-copy update that allows lock free read operations while individual data elements are being updated but does not "preserve group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions." As set forth at the end of the APA, conventional read-copy update can only handle

group integrity by using burdensome techniques that can include read-locking or group copying, which is the problem to be solved by the claimed invention.

The APA also fails to disclose "performi ng a first-phase update operation that preserves a consistent pre-update view of said data element group on behalf of pre-update readers and a consistent post-update view of the data element group on behalf of post-update readers." Again, the APA does not disclose maintaining group consistency while allowing lock free reading, except as a statement of the problem to be solved. The passage at S2:1-8 is referring to pre-update and post-update views of individual data elements, not an entire data element group.

The APA also fails to disclose "providing means by which readers can locate all data elements of said data element group that belong to each of said pre-update and post-update views as readers search said data element group." The conventional read-copy update technique disclosed in the APA does not allow readers to locate all data elements that belong to pre-update and post-update views. The view that is preserved by conventional read-copy update is the data element currently being referenced by the reader. The passage at S2:3-6 merely confirms this aspect of conventional read-copy update.

The APA also fails to disclose "performing one or more read operations following said first-phase update operation in which one or more readers search said data element group with each reader referencing only data elements belonging to one of said pre-update and post-update views." The passage at S2:17-19 merely refers to the fact that conventional read-copy update allows multiple readers to concurrently traverse a linked list without locks.

Based on the foregoing, Applicants submit that the claims in the present application clearly and patentably distinguish over the cited references and it is respectfully requested that the Examiner be reversed and directed to pass the application to issue.

The required Appeal Brief fee is being paid in conjunction herewith.

Respectfully submitted,

/Walter W. Duft/

By: WALTER W. DUFT
Attorney for Applicant(s)
Registration No. 31,948

LAW OFFICES OF WALTER W. DUFT
8616 Main Street, Suite 2
Williamsville, NY 14221
Telephone: (716) 633-1930

CLAIMS APPENDIX

Claim 1 (original): A method for updating a shared data element group while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, comprising:

generating a new group data element;

assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader;

if a prior version of said new data element exists, establishing a version link between said new data element and said prior version;

linking said new data element into said data element group so that it is reachable by readers;

updating a global generation number associated with said data element group; and

if a prior version of said new data element exists, freeing said prior version following a grace period.

Claim 2 (original): A method in accordance with claim 1 wherein said method is used to replace a group data element and said new data element is generated by copying said data element to be replaced.

Claim 3 (original): A method in accordance with claim 1 wherein said method is used to delete a group data element and said new data element is generated by copying said data element to be deleted and setting a deletion flag in said new data element.

Claim 4 (original): A method in accordance with claim 1 wherein said method is used to insert a new group data element and said new data element has no prior versions.

Claim 5 (original): A method in accordance with claim 1 wherein said method further includes generating a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new data element and further being used to link said new data element into said data element group.

Claim 6 (original): A method for updating a shared data element group while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, comprising:

generating a pointer-forwarding entity that points to a data element in said data element group;

assigning a generation number to said pointer-forwarding entity that allows a reader of said data element group to determine whether said pointer-forwarding entity is a correct version for said reader;

if there is a prior version of said pointer-forwarding entity, establishing a version link between said pointer-forwarding entity and said prior version;

linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity;

updating a global generation number associated with said data element group; and

if a prior version of said pointer-forwarding entity exists, freeing said prior version following a grace period.

Claim 7 (original): A method for performing a search of a shared data element group that may be undergoing modification in accordance with the method of claim 1, comprising:

assigning a current global generation number to said search;

when referencing a data element in said data element group, determining whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number; and

searching for a correct version of said referenced data element as necessary.

Claim 8 (original): A method in accordance with Claim 7 wherein, if said data element generation number is equal to said search generation number, said referenced data element is accepted for reading as a correct version.

Claim 9 (original): A method in accordance with Claim 7 wherein, if said data element generation number is less than said search generation number, a search is made for a later version of said referenced data element, and wherein said referenced data element is used if a later version is not found.

Claim 10 (original): A method in accordance with Claim 7 wherein, if said data element generation number is greater than said search generation number, a search is made for a prior version of said referenced data element, and wherein said referenced data element is deemed to be a new insertion if there is no prior version.

Claim 11 (original): A data processing system having one or more central processing units, a memory and a communication pathway between the one or more central processing units and the memory, said system being adapted to update a shared data element group in said memory while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, and comprising:

means for generating a new group data element;

means for assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader;

means for establishing, if a prior version of said new data element exists, a version link between said new data element and said prior version;

means for linking said new data element into said data element group so that it is reachable by readers;

means for updating a global generation number associated with said data element group; and

means for freeing, if a prior version of said new data element exists, said prior version following a grace period.

Claim 12 (original): A system in accordance with claim 11 wherein said system is adapted to replace a group data element and to generate said new data element by copying said data element to be replaced.

Claim 13 (original): A system in accordance with claim 11 wherein said system is adapted to delete a group data element and to generate said new data element by copying said data element to be deleted and setting a deletion flag in said new data element.

Claim 14 (original): A system in accordance with claim 11 wherein said system is adapted to insert a new group data element such that said new data element has no prior versions.

Claim 15 (original): A system in accordance with claim 11 wherein said system further includes means for generating a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new

data element and further being used to link said new data element into said data element group.

Claim 16 (original): A data processing system having one or more central processing units, a memory and a communication pathway between the one or more central processing units and the memory, said system being adapted to update a shared data element group in said memory while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, and comprising:

means for generating a pointer-forwarding entity that points to a data element in said data element group;

means for assigning a generation number to said pointer-forwarding entity that allows a reader of said data element group to determine whether said pointer-forwarding entity is a correct version for said reader;

means for establishing, if there is a prior version of said pointer-forwarding entity, a version link between said pointer-forwarding entity and said prior version;

means for linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity;

means for updating a global generation number associated with said data element group; and

means for freeing, if a prior version of said pointer-forwarding entity exists, said prior version following a grace period.

Claim 17 (original): A data processing system having one or more central processing units, a memory and a communication pathway between the one or more central processing units and

the memory, said system being adapted to perform a search of a shared data element group that may be undergoing modification by the system the system of claim 11, and comprising:

means for assigning a current global generation number to said search;

means for determining, when referencing a data element in said data element group, whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number; and

means for searching for a correct version of said referenced data element as necessary.

Claim 18 (original): A system in accordance with Claim 17 wherein said system is adapted to accept said referenced data element for reading as a correct version if said data element generation number is equal to said search generation number.

Claim 19 (original): A system in accordance with Claim 17 wherein said system is adapted to search for a later version of said referenced data element if said data element generation number is less than said search generation number, and to use said referenced data element if a later version is not found.

Claim 20 (original): A system in accordance with Claim 17 wherein said system is adapted to search for a prior version of said referenced data element if said data element generation number is greater than said search generation number, and to deem said referenced data element to be a new insertion if there is no prior version.

Claim 21 (original): A computer program product for updating a shared data element group while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, comprising:

one or more data storage media;

means recorded on said data storage media for programming a data processing platform to operate as by:

generating a new group data element;

assigning a generation number to said new data element that allows a reader of said data element group to determine whether said new data element is a correct version for said reader;

if a prior version of said new data element exists, establishing a version link between said new data element and said prior version;

linking said new data element into said data element group so that it is reachable by readers;

updating a global generation number associated with said data element group; and

if a prior version of said new data element exists, freeing said prior version following a grace period.

Claim 22 (original): A computer program product in accordance with claim 21 wherein said product is adapted to replace a group data element and to generate said new data element by copying said data element to be replaced.

Claim 23 (original): A computer program product in accordance with claim 21 wherein said product is adapted to delete a group data element and to generate said new data element by copying said data element to be deleted and setting a deletion flag in said new data element.

Claim 24 (original): A computer program product in accordance with claim 21 wherein said product is adapted to insert a new group data element such that said new data element has no prior versions.

Claim 25 (original): A computer program product in accordance with claim 21 wherein said means recorded on said data storage media are further adapted to generate a pointer-forwarding entity that points to said new data element, said pointer forwarding entity maintaining said version link on behalf of said new data element and further being used to link said new data element into said data element group.

Claim 26 (original): A computer program product for updating a shared data element group while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, comprising:

one or more data storage media;

means recorded on said data storage media for programming a data processing platform to operate as by:

generating a pointer-forwarding entity that points to a data element in said data element group;

assigning a generation number to said pointer-forwarding entity that allows a reader of said data element group to determine whether said pointer-forwarding entity is valid for said reader;

if there is a prior version of said pointer-forwarding entity, establishing a version link between said pointer-forwarding entity and said prior version;

linking said pointer-forwarding entity into said data element group so that said data element pointed to by said pointer-forwarding entity is reachable by readers through said pointer-forwarding entity;

updating a global generation number associated with said data element group; and

if a prior version of said pointer-forwarding entity exists, freeing said prior version following a grace period.

Claim 27 (original): A computer program product for performing a search of a shared data element group that may be undergoing modification in accordance with the method of claim 1, comprising:

one or more data storage media;

means recorded on said data storage media for programming a data processing platform to operate as by:

assigning a current global generation number to said search;

when referencing a data element in said data element group, determining whether said referenced data element is a correct version by comparing a generation number assigned to said referenced data element with said search generation number; and

searching for a correct version of said referenced data element as necessary.

Claim 28 (original): A computer program product in accordance with Claim 27 wherein, if said data element generation number is equal to said search generation number, said referenced data element is accepted for reading as a correct version.

Claim 29 (original): A computer program product in accordance with Claim 27 wherein, if said data element generation number is less than said search generation number, a search is made for a later version of said referenced data element, and wherein said referenced data element is used if a later version is not found.

Claim 30 (original): A computer program product in accordance with Claim 27 wherein, if said data element generation number is greater than said search generation number, a search is made for a prior version of said referenced data element, and wherein said referenced data

element is deemed to be a new insertion if there is no prior version.

Claim 31 (original): A computer program product for managing a shared data element group so as to allow updates thereof while preserving group integrity on behalf of one or more readers that are concurrently referencing group data elements without using locks or atomic instructions, comprising:

one or more data storage media;

means recorded on said data storage media for programming a data processing platform to operate as by:

performing a first-phase update operation that preserves a consistent pre-update view of said data element group on behalf of pre-update readers and a consistent post-update view of the data element group on behalf of post-update readers;

providing means by which readers can locate all data elements of said data element group that belong to each of said pre-update and post-update views as readers search said data element group;

performing one or more read operations following said first-phase update operation in which one or more readers search said data element group with each reader referencing only data elements belonging to one of said pre-update and post-update views; and

performing a second-phase update operation following a grace period that frees said pre-update view of said data element group.

EVIDENCE APPENDIX

NONE

RELATED PROCEEDINGS APPENDIX

NONE